Programming Assignment Unit 3

Title: Simple Clock Application
Student: Said HASHEMI

Introduction

Multithreading is one of the most important concepts in Java because it allows multiple tasks to run at the same time, improving responsiveness and efficiency. Instead of making the whole program wait for one task to finish, multithreading lets tasks overlap, which is particularly useful in real-world applications such as clocks, servers, and GUIs (Eck, 2022).

This assignment asked me to build a simple clock application that uses Java threads to continuously update and display the current time. The project helped me practice thread creation, synchronization, and assigning thread priorities, which are key skills for writing efficient concurrent applications.

Source code

```
import java.text.SimpleDateFormat;

import java.util.Date;


// The Clock class is responsible for keeping and updating the current time

class Clock {

    private String currentTime;


    // Method to update the current time continuously

    public void updateTime() {

            SimpleDateFormat formatter = new SimpleDateFormat("HH:mm:ss dd-MM-yyyy");

        while (true) {

            synchronized (this) {

                currentTime = formatter.format(new Date());

            }

            try {

                // Sleep for half a second before updating again

                Thread.sleep(500);
```

```
            } catch (InterruptedException e) {
                System.out.println("Time update interrupted: " + e.getMessage());
            }
        }
    }

    // Method to get the current time
    public synchronized String getCurrentTime() {
        return currentTime;
    }
}

// Thread class to update the clock in the background
class TimeUpdater extends Thread {
    private Clock clock;

    public TimeUpdater(Clock clock) {
        this.clock = clock;
    }

    @Override
    public void run() {
        clock.updateTime();
    }
}

// Thread class to display the clock time
class TimeDisplay extends Thread {
    private Clock clock;

    public TimeDisplay(Clock clock) {
        this.clock = clock;
    }
```

```
    @Override
    public void run() {
        while (true) {
            String time = clock.getCurrentTime();
            if (time != null) {
                System.out.println("Current Time: " + time);
            }
            try {
                // Sleep for one second before printing again
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                System.out.println("Display interrupted: " + e.getMessage());
            }
        }
    }
}


public class Main {
    public static void main(String[] args) {
        Clock clock = new Clock();

        // Create threads
        TimeUpdater updater = new TimeUpdater(clock);
        TimeDisplay display = new TimeDisplay(clock);

        // Assign priorities
        updater.setPriority(Thread.MIN_PRIORITY);   // lower priority
        display.setPriority(Thread.MAX_PRIORITY);   // higher priority

        // Start threads
        updater.start();
        display.start();
    }
```

}

Explanation of the Code

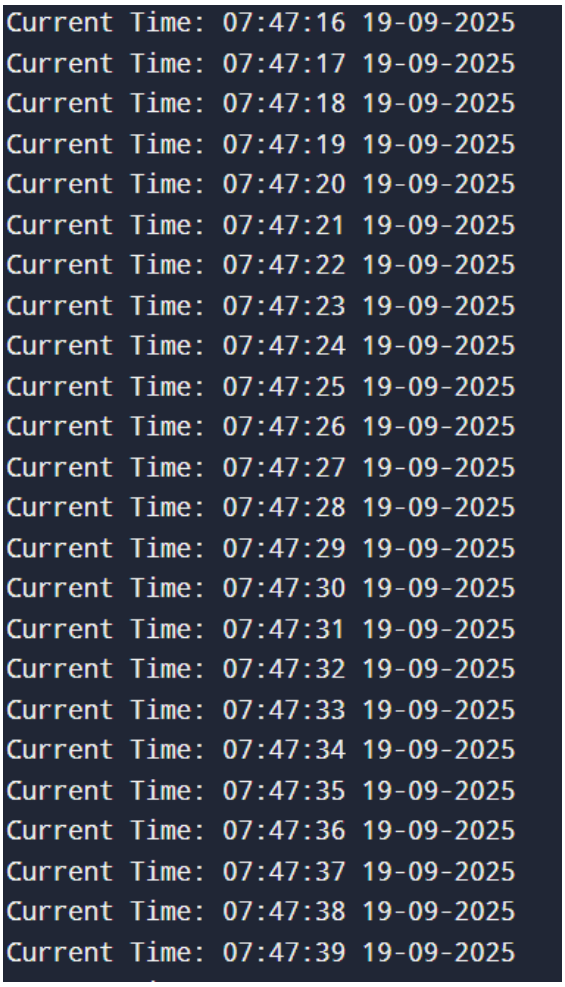
This program is a multithreaded digital clock in Java. It consists of a Clock class that stores and updates the current time using SimpleDateFormat. The TimeUpdater thread continuously updates the time every 0.5 seconds, while the TimeDisplay thread prints the current time to the console every second. The Clock class uses synchronization to ensure thread-safe access to the time. In the Main class, both threads are created, priorities are set, and they are started, resulting in a live, continuously updating clock displayed on the screen.

Multithreading is beneficial because it allows programs to perform multiple actions at once, improving speed and user experience. For example, in this project, if we only had one thread, updating and displaying the time could block each other. With two threads, the application runs smoothly and continuously updates the display.

Threads can be in different states such as **new, runnable, running, waiting, timed waiting, and terminated** (Eck, 2022). In this program, the TimeUpdater and TimeDisplay threads cycle between runnable, running, and timed waiting (when they sleep).

The videos emphasized the difference between creating threads by extending Thread or implementing Runnable (Telusko, 2023d). For simplicity, I extended the Thread class here, but in real projects, using Runnable is often preferred because it supports better object-oriented design.

Output



Conclusion

This project showed me how Java threads can be used to create real-world applications like a simple clock. I learned how to create and manage threads, synchronize data access, and use thread priorities for responsiveness. Multithreading increases efficiency and prevents programs from freezing, which is why it is a fundamental concept in Java programming (Telusko, 2023a).

References

Coding with John. (2021, June 28). *Multithreading in Java explained in 10 minutes* [Video]. YouTube. https://www.youtube.com/watch?v=mtCzN7U0M2g

Eck, D. J. (2022). *Introduction to programming using Java version 9, JavaFX edition.* Licensed under CC BY 4.0. https://math.hws.edu/javanotes/

Samoylov, N. (2018). *Introduction to programming: Learn to program in Java with data structures, algorithms, and logic.* Packt Publishing.

Telusko. (2023a, January 19). *#85 Threads in Java* https://youtu.be/r_MbozD32eo .

Telusko. (2023b, January 19). *#86 Multiple threads in Java* https://youtu.be/KuvkahVyY9E YouTube.

Telusko. (2023c, January 19). *#87 Thread priority and sleep in Java* https://youtu.be/UfMM924sBvg YouTube.

Telusko. (2023d, January 19). *#88 Runnable vs thread in Java* https://youtu.be/yyLy-an_CXY YouTube.